CmoPy: Constrained Multiobjective Optimization in Python

Aljoša Vodopija
Jožef Stefan Institute and
Jožef Stefan International Postgraduate School
Jamova cesta 39
SI-1000 Ljubljana, Slovenia
aljosa.vodopija@ijs.si

Bogdan Filipič
Jožef Stefan Institute and
Jožef Stefan International Postgraduate School
Jamova cesta 39
SI-1000 Ljubljana, Slovenia
bogdan.filipic@ijs.si

ABSTRACT

Python is one of the most frequently used programming languages for solving multiobjective optimization problems (MOPs). Although there exist Python packages covering a wide range of multiobjective optimization tools, there is still a lack of implemented constraint handling techniques (CHTs). This paper introduces a new package for Constrained Multiobjective Optimization in Python (*CmoPy*). It describes the package implementation, the included CHTs and benchmark problems, and shows examples of using the package. With five state-of-the art CHTs and 22 benchmark constrained MOPs, *CmoPy* is currently the most comprehensive Python package for constrained mutliobjective optimization.

Keywords

Python, constrained multiobjective optimization, multiobjective evolutionary algorithms, constraint handling techniques, NSGA-II algorithm

1. INTRODUCTION

Real-world optimization problems regularly involve both multiple objectives and constraints. Such problems are called *constrained multiobjective optimization problems* (CMOPs). A CMOP can be formulated as

$$\begin{aligned} \text{minimize} \quad & f_m(x), \quad m = 1, \ldots, M \\ \text{subject to} \quad & g_n(x) \leq 0, \quad n = 1, \ldots, N \end{aligned}$$

where $x = (x_1, \ldots, x_D)^{\mathrm{T}}$ is a *decision vector* (solution), $f_m : S \to \mathbb{R}$ are *objective functions*, $g_n : S \to \mathbb{R}$ *constraint functions*, $S \subseteq \mathbb{R}^D$ is a *decision space* of dimension D, and M and N are the numbers of objectives and constraints, respectively. Additionally, $f_m(x)$ is an *objective value*, $g_n(x)$ a *constraint value*, $\phi_n(x) = \max(g_n(x), 0)$ *constraint violation* and $\phi(x) = \sum_n \phi_n(x)$ the *overall constraint violation*. A solution satisfying all the constraints is a *feasible solution* and otherwise an *infeasible solution*.

A feasible solution $x^{(1)}$ is said to *dominate* a feasible solution $x^{(2)}$ if and only if $f_m(x^{(1)}) \leq f_m(x^{(2)})$ for all $m \in \{1, \ldots, M\}$ and $f_m(x^{(1)}) < f_m(x^{(2)})$ for at least one index m. A feasible solution x^* is a *Pareto-optimal solution*, if there are no feasible solutions from S dominating x^*. The set of all Pareto-optimal solutions from S is called a *Pareto-optimal set* and its image in the objective space a *Pareto front*. The goal of multiobjective optimization is to find an approximation of the Pareto front that represents trade-offs between the objectives.

Python is one of the most frequently used programming languages for solving multiobjective optimization problems. Due to its simplicity, versatility and accessibility of numerous open source optimization tools, it is suitable for academic research as well as for real-world applications. In the Python Package Index—the biggest repository of Python software—there are currently six packages dealing with (nonlinear) multiobjective optimization problems: *DEAP* [7], *inspyred* [9], *jMetalPy* [1], *PaGMO 2.0* [2], *Platypus* [10] and *pymoo* [3]. All these packages implement the most common evolutionary algorithms for multiobjective optimization, such as Nondominated Sorting Genetic Algorithm II (NSGA-II) [5], Multiobjective Optimization Evolutionary Algorithm Based on Decomposition (MOEA/D) [18] and others.

However, constraint handling techniques (CHTs) are not explicitly addressed in these packages. In *jMetalPy*, *PaGMO 2.0* and *Platypus*, only the constrained-domination principle (CDP) [5] is implemented to deal with constraints. In *pymoo* and *inspyred*, constraints are not considered at all, while *DEAP* implements a CHT based on a penalty function. For this reason, there is still a great need for a more comprehensive package covering the area of constrained multiobjective optimization.

In this paper, we present a new package for constrained multiobjective optimization named *CmoPy*, which stands for Constrained Multiobjective Optimization in Python. The package is designed as a potential functionality in the *SciPy* Python tool [12]. It implements five state-of-the-art CHTs and an ensemble method capable of including multiple CHTs to handle constraints. In addition, several widely used CMOPs are included for benchmarking purposes.

The rest of this paper is organized as follows. Section 2 describes the implementation of *CmoPy*. Section 3 is dedicated to the CHTs implemented in the package, while the included CMOPs are covered in Section 4. In Section 5, some examples of using the package are shown. Finally, Section 6 summarizes the *CmoPy* presentation and provides ideas for future work.

2. PACKAGE IMPLEMENTATION

The main function in *CmoPy* is `nsga_ii`. When called, this function executes the NSGA-II algorithm to solve the given CMOP. The `nsga_ii` function resembles the original NSGA-II [5] in all segments except for minor modifications in the survivor selection phase, where the population is selected for

the new generation, to allow for the inclusion of additional CHTs (see [17] for details on these modifications and Section 3 for a summary of CHTs implemented in *CmoPy*). The input and output parameters of the `nsga_ii` function are described below.

The input parameters of the `nsga_ii` function:

- `problem (Problem)`: A custom object including four parameters:
  - `fun (callable)`: A Python callable (function) consisting of objective and constraint functions. Must be of the form `f(x, *args)` where `x` is the argument in the form of a 1-D `ndarray` and `args` is a tuple of all additional fixed parameters needed to fully specify the objective and/or constraint functions. The output must be a 1-D `ndarray` of objective and constraint values. Note that only inequality constraints of the form $g(x) \leq 0$ are accepted. Equality and other forms of constraints need to be reformulated as inequality constraints.
  - `bounds (list of tuple)`: Bounds of the form `(min, max)` that define the lower and upper bounds for the arguments of `fun`.
  - `no_cons (int, optional)`: The number of constraints. Default is 0.
  - `args (tuple, optional)`: Any additional parameters needed to completely specify the objective and/or constraint functions. Default is `None`.
- `max_iter (int, optional)`: The maximum number of generations to be executed by the optimizer. Default is 250.
- `pop_size (int, optional)`: Number of solutions in the population. Default is 100.
- `max_fun (int, optional)`: The maximum number of function evaluations. Default is 25,000.
- `mut_prob (float, optional)`: The probability for a solution to be mutated. Default is $1/D$.
- `cross_prob (float, optional)`: The probability for parents to be altered by crossover. Default is 0.9.
- `mut_eta (int, optional)`: The distribution index for the polynomial mutation. Default is 20.
- `cross_eta (int, optional)`: The distribution index for the simulated binary crossover. Default is 20.
- `seed (int, optional)`: This parameter controls the seeds of the stochastic processes applied during the algorithm run. If no value is specified, a random seed is used.
- `init (str or ndarray, optional)`: The type of the population initialization. It can be `"lhs"` for the latin hypercube sampling, `"rand"` for random population initialization, or a `ndarray` of predefined solutions. In the last case, the parameter `pop_size` is equal to the number of rows in the array. Default is `"lhs"`.
- `cht (tuple of str, optional)`: The required constraint handling technique. It can be `"nds"` for nondominated sorting, `"cdp"` for constrained-domination principle, `"dpf"` for dynamic penalty function, `"str"` for stochastic ranking, `"mcr"` for multiple constraint ranking, or a tuple of any set of these methods. In the latter case, the ensemble of specified CHTs is used. The default value is `"nds"` if there are no constraints and `"cdp"` otherwise.

The output parameter is a custom object named `Result`. It is a multiobjective extension of the object `OptimizeResult` used to represent results in the optimization module of *SciPy* [12]. The object `Result` includes the following parameters:

- `x (ndarray)`: Solutions from the final population.
- `x_all (ndarray)`: All nondominated feasible solutions found during the entire optimization run.
- `success (bool)`: Whether or not the optimizer exited successfully.
- `status (int)`: The optimizer termination status.
- `message (str)`: The description of the termination cause.
- `fun (ndarray)`: The objective and constraint values of solutions from the final population.
- `fun_all (ndarray)`: The objective and constraint values of all nondominated feasible solutions found during the entire run.
- `nfev (int)`: The number of the `fun` function evaluations.
- `nit (int)`: The number of generations executed by the optimizer.
- `maxcv (float)`: The maximum overall constraint violation.

The implementation of *CmoPy* follows the guidelines for contributing to *SciPy*. The only dependency apart from *SciPy* needed to use the package is *NumPy* [14].

3. CONSTRAINT HANDLING TECHNIQUES

In *CmoPy*, there are five widely used CHTs and an ensemble method combining any desired set of single techniques:

- Nondominated sorting [5]: This method selects the new generation of solutions according to the dominance relation, not considering constraint violations at all. This method is used as the default CHT for unconstrained problems.
- Constrained-domination principle [5]: This CHT can be seen as an extension of the nondominated sorting, where feasible solutions dominate infeasible ones, and infeasible solutions are ranked according to the overall constraint violation. This method is used as the default CHT for constrained problems.

- Dynamic penalty function [6]: This method augments the fitness of a solution by adding a penalty that is proportional to the overall constraint violation. The penalty pressure is increased in each generation.
- Stochastic ranking [15]: This CHT uses a bubble-sort-like process to rank solutions in the population. Two feasible solutions are always compared based on their fitness. On the other hand, if at least one of the solutions is infeasible, then a random decision is made on whether the two solutions are compared based on their fitness or constraint violation.
- Multiple constraint ranking [8]: In this approach, the solutions are ranked based on their fitness and constraint violation. If there are no feasible solutions, only the rank generated from constraint violation is considered, otherwise a combination of both ranks is taken into account.
- Ensemble of CHTs [17]: This approach combines multiple single CHTs into an ensemble-based ranking. The solutions are ranked based on a quality measure which is averaged over all techniques in the ensemble.

4. INCLUDED BENCHMARK PROBLEMS

The *CmoPy* package contains 22 CMOPs that are frequently used for benchmarking purposes (see, for example, [4, 16]). Table 1 summarizes these problems considering three basic characteristics: the dimension of the decision space, the number of objectives and the number of constraints.

Table 1: Characteristics of the CMOPs included in *CmoPy*: dimension of the decision space D, number of objectives M and number of constraints N.

CMOP	D	M	N
Belegundu [4]	2	2	2
Binh 1 [4]	2	2	2
Binh 2 [4]	2	3	2
C1-DTLZ1 [11]	$M+4$	≥ 2	1
C1-DTLZ3 [11]	$M+9$	≥ 2	1
C2-DTLZ2 [11]	$M+9$	≥ 2	1
C3-DTLZ1 [11]	$M+4$	≥ 2	M
C3-DTLZ4 [11]	$M+4$	≥ 2	M
Car-side impact [11]	3	10	7
DTLZ8 [4]	30	3	3
DTLZ9 [4]	30	3	2
Jimenez [4]	2	2	4
Kita [4]	2	2	3
Obayashi [4]	2	2	1
Osyczka 1 [4]	2	2	2
Osyczka 2 [4]	6	2	6
Srinivas [4]	2	2	2
Tamaki [4]	3	3	1
Tanaka [4]	2	2	2
Vibrating platform [13]	2	5	5
Viennet [4]	2	3	2
Water resource planning [11]	5	7	3

5. EXAMPLES OF USING THE PACKAGE

This section illustrates the use of *CmoPy* on the well-known Srinivas [4] problem. It also shows an example of adding a new CMOP to the package.

We first need to install *CmoPy*. This can be achieved by running the following commands:

```
$ git clone https://gitlab.com/cmopy/cmopy.git
$ cd cmopy
$ python setup.py
```

In the following example, the NSGA-II algorithm is run to solve the Srinivas problem. The population size is set to 80 solutions (`pop_size=80`), while the number of generations is kept at the default value of 250. In addition, CDP is used for handling constraints as a default option.

```
>>> from cmopy.optimize import nsga_ii
>>> from cmopy.problems import srinivas, Problem
>>> nsga_ii(srinivas, pop_size=80)
```

To add new CMOPs to *CmoPy*, we need to implement the `problem` object introduced in Section 2. In addition to the `callable` object (`test_fun`) consisting of the objective and constraint functions, we need to specify the bounds (`bounds`) and the number of constraints (`no_cons`). The following example shows the implementation of the Tanaka problem [4].

```
>>> import numpy as np
>>> def test_fun(x):
...     x1, x2 = x
...     g1 = 1 - x1 ** 2 - x2 ** 2 + \
...          0.1 * np.cos(16 * np.arctan(x1 / x2))
...     g2 = (x1 - 0.5) ** 2 + \
...          (x2 - 0.5) ** 2 - 0.5
...     return np.array([x1, x2, g1, g2])
>>> bounds = [(0, np.pi)] * 2
>>> no_cons = 2
>>> test_cmop = Problem(test_fun, bounds, no_cons)
```

At this point, we can run the NSGA-II algorithm to solve the implemented problem. Here, we use the dynamic penalty function to handle constraints (`cht="dpf"`).

```
>>> nsga_ii(test_cmop, cht="dpf")
```

If we want to run an ensemble combining multiple CHTs, we need to specify all the desired techniques in a tuple. The `nsga_ii` function automatically detects that an ensemble method is required. In the following example, the optimizer uses an ensemble combining the constrained-domination principle and the dynamic penalty function to handle constraints (`cht=("cdp", "dpf")`).

```
>>> nsga_ii(test_cmop, cht=("cdp", "dpf"))
```

Figure 1 shows the Pareto front approximations for the Srinivas problem (left) and the Tanaka problem (right) obtained after running the above commands.

6. CONCLUSIONS

We introduced *CmoPy*, a Python package designed for constrained multiobjective optimization. Unlike other Python packages for multiobjective optimization, it contains a comprehensive set of CHTs. While other packages usually implement one simple method to deal with constraints, in *CmoPy* there are five state-off-the-art CHTs and an ensemble-based

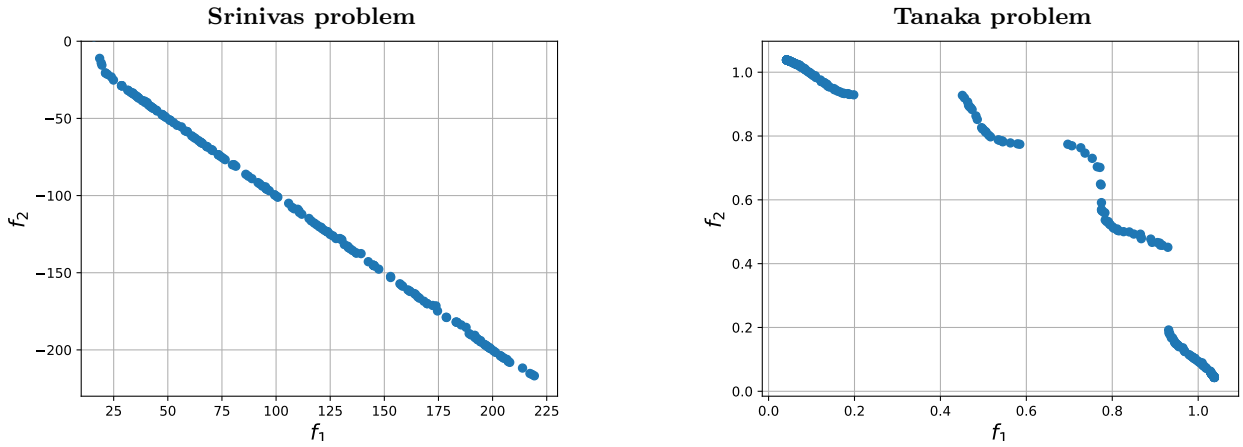

Figure 1: Pareto front approximations generated by *CmoPy* for the Srinivas problem (left) and the Tanaka problem (right). The former was solved by handling constraints with CDP, while the latter by applying an ensemble of CHTs.

method combining multiple single techniques. Moreover, an extensive set of benchmark CMOPs with various characteristics is also included in *CmoPy*.

In the future, we plan to extend the package functionality by adding other multiobjective optimizers, for example, MOEA/D, and integrating the methods for visualizing the results and algorithm performance.

7. ACKNOWLEDGMENTS

The authors acknowledge the financial support from the Slovenian Research Agency (Young researcher program and Research core funding No. P2-0209).

8. REFERENCES

[1] A. Benítez-Hidalgo, A. J. Nebro, J. García-Nieto, I. Oregi, and J. D. Ser. jMetalPy: A Python framework for multi-objective optimization with metaheuristics. http://arxiv.org/abs/1903.02915, 2019.

[2] F. Biscani et al. PaGMO 2.0. https://github.com/esa/pagmo2, 2019.

[3] J. Blank and K. Deb. pymoo: Multi-objective optimization in Python. https://pymoo.org, 2019.

[4] C. A. Coello Coello, G. B. Lamont, and D. A. Van Veldhuizen. *Evolutionary Algorithms for Solving Multi-Objective Problems*, chapter 4, pages 179–238. Springer-Verlag Berlin Heidelberg, 2nd edition, 2007.

[5] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197, 2002.

[6] A. E. Eiben and J. E. Smith. *Introduction to Evolutionary Computing*, chapter 8, pages 129–151. Natural Computing Series. Springer-Verlag Berlin Heidelberg, 2003.

[7] F.-A. Fortin, F.-M. De Rainville, M.-A. Gardner, M. Parizeau, and C. Gagné. DEAP: Evolutionary algorithms made easy. *Journal of Machine Learning Research*, 13(1):2171–2175, 2012.

[8] R. D. P. Garcia, B. S. L. P. de Lima, A. C. D. C. Lemonge, and B. P. Jacob. A rank-based constraint handling technique for engineering design optimization problems solved by genetic algorithms. *Computers & Structures*, 187:77–87, 2017.

[9] A. Garrett. inspyred: Bio-inspired algorithms in Python. https://github.com/aarongarrett/inspyred, 2019.

[10] D. Hadka. Platypus: Multiobjective optimization in Python. https://github.com/Project-Platypus/Platypus, 2019.

[11] H. Jain and K. Deb. An evolutionary many-objective optimization algorithm using reference-point based nondominated sorting approach, Part II: Handling constraints and extending to an adaptive approach. *IEEE Transactions on Evolutionary Computation*, 18(4):602–622, 2014.

[12] E. Jones, T. Oliphant, and P. Peterson. SciPy: Open source scientific tools for Python. http://www.scipy.org, 2019.

[13] A. Messac. Physical programming: Effective optimization for computational design. *American Institute of Aeronautics and Astronautics Journal*, 34(1):149–158, 1996.

[14] T. E. Oliphant. *A guide to NumPy*, volume 1. Trelgol Publishing USA, 2006.

[15] T. P. Runarsson and X. Yao. Stochastic ranking for constrained evolutionary optimization. *IEEE Transactions on Evolutionary Computation*, 4(3):284–294, 2000.

[16] R. Tanabe and A. Oyama. A note on constrained multi-objective optimization benchmark problems. In *2017 IEEE Congress on Evolutionary Computation*, CEC 2017, pages 1127–1134. IEEE, 2017.

[17] A. Vodopija, A. Oyama, and B. Filipič. Ensemble-based constraint handling in multiobjective optimization. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, GECCO '19, pages 2072–2075. ACM, 2019.

[18] Q. Zhang and H. Li. MOEA/D: A multiobjective evolutionary algorithm based on decomposition. *IEEE Transactions on Evolutionary Computation*, 11(6):712–731, 2007.